



PATENT ABSTRACTS OF JAPAN

(11) Publication number: **01046840 A**(43) Date of publication of application: **21.02.89**

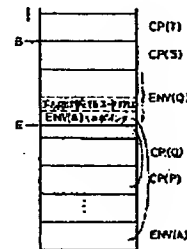
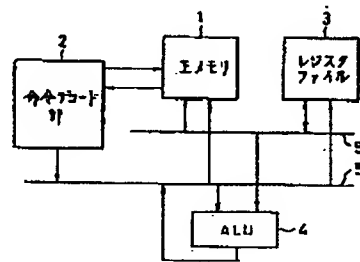
(51) Int. Cl.

G06F 9/44(21) Application number: **62203712**(71) Applicant: **TOSHIBA CORP**(22) Date of filing: **17.08.87**(72) Inventor: **KOYAMA KIYOMI****(54) PROLOG PROCESSOR****(57) Abstract:**

PURPOSE: To increase the prolog processing speed by deciding the goal of a main body part when a prolog language is compiled and omitting the relevant processing when no allocation of a back track point is required to a local stack.

CONSTITUTION: The titled processor is composed of a main memory 1, an instruction decoding part 2, a register file 3 and an arithmetic and logic unit ALU4. Then it is checked whether a selection point frame CP is produced or not when a prolog language is compiled and the goal of a main body part is carried out. If not, a process where a back track point is allocated to a local stack can be omitted. Thus the memory access for said allocation is eliminated and the prolog processing speed is increased.

COPYRIGHT: (C)1989,JPO&Japio



⑩ 日本国特許庁 (JP)

⑪ 特許出願公開

⑫ 公開特許公報 (A)

昭64-46840

⑬ Int. Cl.⁴

G 06 F 9/44

識別記号

3 3 0

庁内整理番号

B-8724-5B

⑭ 公開 昭和64年(1989)2月21日

審査請求 未請求 発明の数 1 (全13頁)

⑮ 発明の名称 PROLOG処理装置

⑯ 特 願 昭62-203712

⑰ 出 願 昭62(1987)8月17日

⑱ 発 明 者 小 山 清 美 神奈川県川崎市幸区小向東芝町1番地 株式会社東芝総合
研究所内

⑲ 出 願 人 株 式 会 社 東 芝 神奈川県川崎市幸区堀川町72番地

⑳ 代 理 人 弁 理 士 鈴 江 武 彦 外2名

明 細 書

1. 発明の名称

PROLOG処理装置

2. 特許請求の範囲

(1) PROLOGプログラムを解釈し、同一のローカルスタック上に環境フレーム、コンティニュエーションフレーム及び選択点フレームを形成しながらプログラムの実行を進めるための上記プログラムに対応したマシン命令を生成するコンパイラと、このコンパイラによって生成された前記命令コードを記憶するとともに前記ローカルスタックを提供するメモリと、このメモリに記憶されたマシン命令を実行する命令実行部とを備えたPROLOG処理装置において、前記コンパイラは、サブゴールを起動する必要のないゴールをリストアップしたテーブルと、前記プログラム中にカットオペレータを含む節が存在するとき、その節の実行中に新たに選択点フレームを生成する可能性の有無を前記テーブルに基づき判定する第1の判定手段と、この第1の判定手段により新たな選択

点フレームが生成されると判定された場合には、その節に続く同一の頭部を持つ他の節が存在するかどうかを判定する第2の判定手段と、この第2の判定手段により他の節が存在すると判定された場合には前記ローカルスタックのトップに現在の選択点フレームよりも一つ手前の前選択点フレームへのポインタをバックトラックポイントとして格納する第1のマシン命令を生成し当該節の環境フレームを形成するマシン命令の前に挿入する第1のマシン命令生成手段と、前記第2の判定手段によって前記他の節が存在しないと判定された場合には前記ローカルスタックのトップに現在の選択点フレームへのポインタをバックトラックポイントとして格納する第2のマシン命令を生成し当該節の環境フレームを形成するマシン命令の前に挿入する第2のマシン命令生成手段と、上記第1及び第2のマシン命令生成手段によるマシン命令の生成が行われた場合に現環境フレームへのポインタから所定のオフセット量を加えたローカルスタック上の位置からデータを読出して現選択点フ

フレームのポインタ用のレジスタにロードする第3のマシン命令を生成しカットオペレータを実行するマシン命令の位置に挿入する第3のマシン命令生成手段と、前記第1の判定手段によって新たな選択点フレームが生成されないと判定された場合には、その節に続く同一の頭部を持つ他の節が存在するかどうかを判定する第3の判定手段と、この第3の判定手段によって他の節が存在すると判定されたときに、前選択点フレームへのポインタを前記ローカルスタックから読み出して現選択点フレームのポインタ用のレジスタにロードする第4のマシン命令を生成しカットオペレータを実行するマシン命令の位置に挿入する第4のマシン命令生成手段とを具備したものであることを特徴とするPROLOG処理装置。

(2)前記コンパイラは、前記カットオペレータを含まない節に対しても、その節の実行中に新たな選択点フレームを生成する可能性の有無を前記テーブルに基づいて判定し、新たな選択点フレームが生成されないと判定された場合には、前記環

は「B、C、Dが全て成立すればAは真である」である。

プログラムの実行過程で、この節が後述する一時変数(temporary variable)のみ持つ場合頭部及び本体のゴール間での引数の値の引渡しはレジスタを通して行われ、この節が後述する永久変数(permanent variable)を持つ場合には、ローカルスタックに変数の値セル(value cell)が形成されここを通して変数の値の引渡しが行われる。この引渡しの為にローカルスタックに形成されるフレームを環境(environment)フレーム(ENV)と呼び、文献D. H. D. Warren, "An Abstract PROLOG Instruction Set", AI Center, SRI International, August 1983 (以下文献1と呼ぶ)によれば第9図に示すような構成である。即ち、まず初めにこのENVの直前にローカルスタック上に割付けられた前ENVへのポインタ(継続環境)が形成され、次いで本節の実行が成功に終わった後で次に実行すべき親ゴールに対応するコードアドレス(継続

環境フレーム又は前記コンティニュエーションフレームの生成を行わないマシン命令を生成することを特徴とする特許請求の範囲第1項記載のPROLOG処理装置。

3. 発明の詳細な説明

[発明の目的]

(産業上の利用分野)

本発明は論理型言語であるPROLOGで記述されたプログラムを実行するPROLOG処理装置に関するものである。

(従来の技術)

PROLOG言語で書かれたプログラムは一般に

A : - B, C, D ... ①

の形をとる。:-記号の左辺にあるAを頭部(head)、:-記号の右辺を本体(body)と呼ぶ。本体に含まれるB、C、Dをゴール(goal)と呼び、また全体を節(clause)と呼ぶ。上記①の節の意味するところは、「Aが真であるためには、B、C、Dが真でなければならない」、換言すれ

コード)が形成され、その後、本節の永久変数の値セルが形成される。また当該節と同じ頭部を持つ他の節がプロセッサ中に含まれ、このプロセッサが親ゴールから呼ばれた場合、親ゴールから渡される引数や、プロセッサ実行直前のプロセッサ状態を格納するためにローカルスタック上にフレームが形成される。これを選択点(choice point)フレーム(CP)と呼び文献1によれば第10図に示すような構成となる。このフレームにはPROLOGの実行において必要となる他のスタックへのポインタ、例えばヒープポインタ(H)、トレイルポインタ(TR)などの外、このCPの直前にローカルスタック上に生成されたCPへのポインタ(B)、ある節の実行が失敗(fail)してバックトラックが発生した時に、代わりに実行すべき節に対応するコードアドレスなどが格納され、更に親ゴールから渡される引数(ゴール引数1～ゴール引数M)が格納される。

ところである節が永久変数を持つか否かの判定法は、文献1に示されている。これによれば、当

該節が本体部に2つ以上のゴールを持ち、ある変数が頭部および本体部の第1ゴールのうちの少なくとも一方に現われた場合を1回とカウントし、以後現われるその変数を2回からカウントすると、2回以上現われた変数を永久変数と呼ぶ。永久変数以外の変数が一時変数である。永久変数を持つ節では、既に述べたようにローカルスタックにENVが割付けられる。更に文献1の標準によれば、ある節の本体部のゴール数が2つ以上であって当該節が永久変数を持たない場合、コンティニューエーション(CONT)と呼ばれるフレームをローカルスタックに割付ける。CONTはENVから永久変数の値セルを除いた部分、即ち第9図の継続環境のスロットと継続コードのスロットとで構成され、ENVと同様に現ENVポインタ(エレジスタ)でローカルスタック上での位置が示される。

先にあげた①の節が永久変数を持つと仮定すると、この節に対応するコンパイルコードは第11図のようになる。本図で1行目のallocate命令は

実行する直前にローカルスタックに割付けられ、また本体部の最終ゴール実行直前にローカルスタックから取除かれる。

次にPROLOGのプログラム実行中にENVやCPがローカルスタックに生成、棄却される様子を示す。例として第12図(a)のプログラムを使う。この図で節1が最上位のゴールであり、本体部にP、Q、Rのゴールを含んでいる。節2、3はゴールPに対応するプロセッサを構成し、節4、5はゴールQに対応するプロセッサを構成する。また節4のゴールSに対応して節6、7、8がプロセッサを構成し、ゴールTに対応して節9、10がプロセッサを構成する。ここで、節1、2、4、6、9を実行する場合を考える。節4は永久変数を持たず、節9が永久変数を持つと仮定する。この時のローカルスタックの動きを第12図(b)に示す。図でENV(A)はゴールAのENV、CP(P)はプロセッサPのCP、CONT(Q)はゴールQのCONTを示している。

先ず節1の実行直前にENV(A)が、ローカ

ルスタックにENVを割付ける機能を持ち、8行目のdeallocate命令は逆にローカルスタックからENVを取除く命令である。更に4行目および6行目のcall命令はリターンポインタをセットした上で、オペランドのプロセッサに実行を移す命令である。また9行目のexecute命令では、call命令と同様にオペランドのプロセッサに実行を移すが、リターンポインタのセットは行わず、直前のdeallocate命令実行時にENVからリストアした継続コードをリターンポインタとして使う。2行目のget-args-of命令および3、5、7行目のput-args-of命令は擬似マシン命令である。get-args-of命令は親ゴールからアーギュメントレジスタを介して渡された引数と節の引数とのマッチングをとるget命令の総称で、またput-args-of命令は本体部のゴールの引数をアーギュメントレジスタにロードするput命令の総称である。各マシン命令の詳細動作は文献1に詳しいので、ここではこれ以上の説明を省略する。第11図から明らかな様にENVは永久変数を含む節を

ルスタックに形成され、次いで節2でPのプロセッサが別解を持つためCP(Q)がローカルスタックに形成され、次に節4で同じくQのプロセッサが別解を持つためCP(Q)が、ローカルスタックに形成される。節4では本体部に3個のゴールがありしかも永久変数を持たないため、ローカルスタックにはCP(Q)に次いでCONT(Q)が形成される。節4のゴールS、Tを実行する過程でプロセッサSとTに対応して各々CP(S)、CP(T)が形成される。節9が永久変数を持つために更にENV(T)が形成される。図に示されるようにENV(T)の継続コードの位置には節4のゴールUに対応するコードアドレスが格納され、またENV(T)の継続環境の位置にはCONT(Q)へのポインタが格納される。従って節9の実行でゴールV、ゴールWと成功して、次にゴールXの実行が失敗(fail)してバックトラックが起こると、CP(T)に格納された代替節のアドレス、即ち節10に実行が移るが、ゴールXの実行が成立すると、既に第11図で述べた様

にゴールXの実行に先立ち、ENV(T)の接続コード、継続環境の情報がレジスタにリストアされENV(T)がCONT(Q)まで巻戻されているため、節4のゴールUに実行が移ることになる。節4に対応するCONT(Q)にしても以上と同様で、ゴールUの実行が成功に終わると、CONT(Q)からENV(A)まで巻戻され、節1のゴールRに実行が移る、という形で節1全体の実行が推移する。

この様にPROLOGプログラムを実行する過程でローカルスタックにCPやENV、CONTのフレームを頻繁に形成することが必要となる。

更に次には本体部にカットオペレータが含まれる場合、ローカルスタックにこれらのフレームがどう形成され、棄却されるかを示す。例として次の2つの節からなるPROLOGプログラムを考える。

A: -B, !, C

) ②

A: -D

- Verlag (以下文献2と呼ぶ) に示されるように、実行速度に於いて両者は大きく食い違う可能性を持っている。即ち、ゴールBが非常に複雑で実行に多くの時間を費やすとした場合、プログラム②ではゴールBの成功/不成功を1回だけ確かめれば済むのに対し、プログラム③では、2回必要となる。プログラム③の方が実行により多くの時間を費やし、カットオペレータを使ったプログラム②の方が実行効率が良い。

また、カットオペレータはローカルスタック(メモリ)の使用効率を向上させる効果もある。これを第13図(a)、(b)を使って説明する。第13図(a)のプログラムにおいて、節1を実行する場合を考える。節1のゴールPを実行するためまず節2を実行すると、この節には別解があるため第13図(b)のCP(P)がローカルスタックに生成される。節1のゴールQを実行する場合も、節4を実行すると、同様に第13図(b)のCP(Q)が生成される。ここで節4が永久変数を持つとすると、第13図(b)のENV(Q)

このプログラムの第1節に本体部でゴールBとゴールCの間にカットオペレータが含まれている。上記②のプログラムの意味するところは、ゴールBの実行が成功した場合には、ゴールCの実行に移るが、この場合他の同じ頭部を持つ節の実行を放棄する。従って、もしゴールCが成功すれば節1全体の実行が成立するが、仮りにゴールCの実行が失敗に終わっても節2の実行は試されない。即ち節2の実行が試されるのは節1でカットオペレータを左から右へ通過する以前、言い換えれば節1のゴールBが不成功に終わる場合のみである。このことからプログラム②はカットオペレータを使わずに次の様に書くこともできる。

A: -B, C

) ③

A: -not(B), D

しかし乍ら、上記2つのプログラム②、③は一般的に同一のセマンティックスで動作すると考えることができるものの、文献W. F. Clocksin, et al "Programming in Prolog" Springer

が生成される。本体部のゴールR、Sを実行するため、節6、節8を実行すると、共に別解を持つため、第13図(b)のようにCP(R)、CP(S)がローカルスタックに生成される。さて節4でゴールSの成功の後カットオペレータを左から右へ通過すると、この時点で仮りにその後ゴールTの実行が不成功に終わったとしても、バックトラックしてSの別解を求めるために節9を試したり、またRの別解を求めるために節7を試したりすることが不要となる。また既に述べたようにゴールTの実行が不成功に終わった場合、Qの別解を求めるために節5が試されることもない。以上の事情から、節4の本体部でカットオペレータを左から右へ通過した時点で第13図(b)のローカルスタックでCP(S)、CP(R)が不要となり、更にゴールTが不成功の場合、第13図(b)のENV(Q)、CP(Q)も不要となる。この様にカットオペレータによってそれ以後の実行で不要なフレームが明確になり、これをローカルスタックから順次取除くことによって効率良く

ローカルスタックが使用されることになる。

以上述べた様にカットオペレータを使うことによってプログラムの実行が高速化され、またローカルスタックの使用効率が向上するという利点が見られるが、PROLOGプログラムを実行するマシン上でカットオペレータをどう実現するかが大きな課題となる。即ち、通常CPやENVなどローカルスタック上に形成されるフレームは第11図に示したような明確なマシン命令によって生成、廃棄が行なわれるが、カットオペレータについてはそれに対応するマシン命令は明確に規定されていない。しかも以上の説明でも明らかな様に、カットオペレータを使用した場合のローカルスタックからフレームを解放する状況は、あるCPから直前のCPに巻戻す、或いはあるENVから直前のENVに巻戻すという単純な規則には従わず、不特定多数箇のフレームを飛び越して前に戻るということが発生するため、巻戻す位置をどう効率よく見つけるか、どの時点で巻き戻せば副作用を発生させずに済むかといった点でカットオペ

レータが高速で処理できるインプリメンテーションを考えることは非常に重要な課題になっている。

(発明が解決しようとする問題点)

このように、従来、PROLOGプログラムを高速で実行したり、メモリの使用効率を高める上でカットオペレータが重要な役割を担っているにも拘らず、実際にカットオペレータをPROLOGマシンにインプリメントして効率良く処理することが困難であるという問題があった。

本発明は、このような事情に鑑みてなされたもので、カットオペレータを効率良く処理してPROLOGプログラムの高速実行を可能とするPROLOG処理装置を提供することを目的とする。

更に本発明はカットオペレータを含まない場合も、ローカルスタックへのフレーム生成や廃棄といったコストの高い処理を最少限にとどめて高速実行が可能なPROLOG処理装置を提供することを目的とする。

[発明の構成]

(問題点を解決するための手段)

本発明は、PROLOGプログラムを解釈し、同一のローカルスタック上に環境フレーム(ENV)、コンティニュエーションフレーム(CONT)及び選択点フレーム(CP)を形成しながらプログラムの実行を進めるための上記プログラムに対応したマシン命令を生成するコンパイラと、このコンパイラによって生成された前記命令コードを記憶するとともに前記ローカルスタックを提供するメモリと、このメモリに記憶されたマシン命令を実行する命令実行部とを備えたPROLOG処理装置において、前記コンパイラを次のように構成したことを特徴としている。

即ち、前記コンパイラは、サブゴールを起動する必要のないゴールをリストアップしたテーブルと、前記プログラム中にカットオペレータを含む節が存在するとき、その節の実行中に新たにCPを生成する可能性の有無を前記テーブルに基づき判定する第1の判定手段と、この第1の判定手段により新たなCPが生成されると判定された場合には、その節に続く同一の頭部を持つ他の節が存

在するかどうか(別解を持つかどうか)を判定する第2の判定手段と、この第2の判定手段により他の節が存在すると判定された場合には前記ローカルスタックのトップ(TOS)に現在のCPよりも一つ手前の前CPへのポインタをバックトラックポイントとして格納する第1のマシン命令を生成し当該節のENVを形成するマシン命令の前に挿入する第1のマシン命令生成手段と、前記第2の判定手段によって前記他の節が存在しないと判定された場合には前記TOSに現在のCPへのポインタをバックトラックポイントとして格納する第2のマシン命令を生成し当該節のENVを形成するマシン命令の前に挿入する第2のマシン命令生成手段と、上記第1及び第2のマシン命令生成手段によるマシン命令の生成が行われた場合にENVへのポインタから所定のオフセット量を加えたローカルスタック上の位置からデータを読出して現CPへのポインタ用のレジスタ(以下Bレジスタと呼ぶ)にロードする第3のマシン命令を生成しカットオペレータを実行するマシン命令の

位置に挿入する第3のマシン命令生成手段と、前記第1の判定手段によって新たなCPが生成されないと判定された場合には、その節に続く同一の頭部を持つ他の節が存在するかどうか（別解を持つかどうか）を判定する第3の判定手段と、この第3の判定手段によって他の節が存在すると判定されたときに、前CPへのポインタを前記ローカルスタックから読み出してBレジスタにロードする第4のマシン命令を生成しカットオペレータを実行するマシン命令の位置に挿入する第4のマシン命令生成手段とを具備したものである。

（作 用）

カットオペレータを含む節で、実行中新たにCPを生成しない場合には、カットオペレータを左から右へ通過後、バックトラックが起きた場合のバックトラックポイントは、CPのポインタ内容が当該節の実行開始時と変わらないため、現在のCPポインタを使って得ることができる。即ち、当該節に対応するプロセッサにCPがある場合（別解がある場合）はBレジスタで指示されるC

Pに含まれる前CPへのポインタをBレジスタに戻せばよいし、またCPが無い場合（別解が無い場合）は現在のBレジスタの値がそのままバックトラックポイントになる。従って、何れの場合も新たにローカルスタックにバックトラックポイントを格納する必要が無いし、通常のバックトラックと同じBレジスタをターゲットにしてカットオペレータのバックトラックが実行できる。

逆にカットオペレータを含む節で、実行中新たにCPを生成する可能性がある場合は、当該節のゴールを実行する過程でバックトラックポイントが他のENVやCPに埋もれてしまう。このため当該節を実行開始直後にローカルスタックにバックトラックポイントを格納しておく。即ち、当該節に対応するプロセッサにCPがある場合は前CPへのポインタを、またCPが無い場合は現CPへのポインタを格納しておき、カットオペレータを左から右へ実行する時に、Bレジスタにバックトラックポイントに戻しておく。既に述べたように、ENVポインタからのオフセットで、このバ

ックトラックポイントがアクセスできるように、文献1で述べられた規則とは別に本体部のゴールが1の時もCONTをローカルスタックに割付けておく。このようにして、カットオペレータの右側のゴールで失敗してカットオペレータのバックトラックが起きた場合、通常のバックトラック処理でカットオペレータが処理できる。

更に本体部にカットオペレータを含むか否かに拘らず、本体部のカットオペレータ以外のゴールが、実行中にサブゴールの起動を行わず実行できる述語である場合、文献1の規準とは拘りなく、ローカルスタックにENVやCONTのフレームを生成しないで実行する。

本発明によればコンパイル時に本体部のゴール実行中にCPが生成されるか否かチェックし、されない場合、ローカルスタックにバックトラックポイントを割付ける処理が省けるので、この割付けのためのメモリアクセスが減らせ、スピードアップの効果が得られる。また、従来の規則によらない新しい規則でローカルスタックへのCONT

フレームの割付けを行なうことで、バックトラックポイントをローカルスタックに格納した場合、必ず現ENVポインタのオフセットで取出せる利点がある。しかも、僅か4種の命令をENV形成命令の前か、カットオペレータの位置に生成すれば良く、コンパイルが簡単になる。また本体部実行中に新たにCPを生成するか否かをコンパイル時にチェックして、不必要なフレームの生成を極力減らしており、この面でも大幅なスピードアップ効果が期待できる。

（実施例）

以下、本発明を図示の実施例に基づいて詳細に説明する。

第1図は、本実施例に係るPROLOG処理装置の構成を示す図である。

この装置は、主メモリ1、命令デコード部2、レジスタファイル3及び算術演算ユニット（ALU）4で構成されている。主メモリ1は、PROLOGプログラムを格納する領域、上記PROLOGプログラムからマシン命令を生成するための

コンパイラを格納する領域、このコンパイラによって生成されたマシン命令を格納する領域、ローカルスタック領域、ヒープ領域、トレイル領域等を提供するメモリである。命令デコード部2は、上記主メモリ1に格納されたマシン命令を解釈する。このマシン命令に従って主メモリ1及びレジスタファイル3に格納されたデータがデータバス5を介してALU4に送られ、適宜演算処理を施されてレジスタファイル3や主メモリ1に格納される。

上記コンパイラは、与えられたPROLOGプログラムに対し、これに対応するマシン命令を形成する。もし、与えられたPROLOGプログラムにカットオペレータ(1)を含む節が含まれている場合には、その節については、第2図に示すような処理が行われる。

先ずマシン命令として次の4つの機能を持ったものを追加する。第1はローカルスタックのトップオブスタック(TOS)に現在のCPより1つ手前(ローカルスタックに割付けられた順番で)

のCPへのポインタを格納してローカルスタックポインタの値を1増す命令(第1の命令)で、これを仮りにmark-prv-cp命令と呼ぶ。第2はローカルスタックのTOSに現在のCPへのポインタ(即ちBレジスタの内容)を格納してローカルスタックポインタの値を1増す命令(第2の命令)で、これを仮りにmark-cur-cp命令と呼ぶ。第3は現在のENVへのポインタ(即ちEレジスタの内容)位置からあるオフセット(仮りに-1)を加えた位置から読み出して、Bレジスタにロードする命令(第3の命令)で、これを仮りにcut命令と呼ぶ。第4は現在のCPより1つ手前のCPへのポインタをローカルスタックから格納してBレジスタにロードする命令(第4の命令)で、これを仮りにdcut命令と呼ぶ。これら4つの命令を使ってコンパイルコードを次の手順で生成して行く。

PROLOGの述語の中で、数値演算、論理演算、比較演算その他の組み込み述語は、サブゴールを起動することなく実行できる。これらの述語を

予め分類しておき、コンパイル時にコンパイラが参照できるような形、例えばテーブルにしておく。この分類分けされた述語をもとに、コンパイラは、カットオペレータを含む当該節が新たにCPを生成するか判定し(S1)、CPを生成する可能性が無いと判定された場合は更に、この節を含むプロセッサのCPがあるか否かを判定する(S6)。CPが無い場合は何もしないが、CPが有る場合は、カットオペレータ位置にdcut命令を生成する(S7)。一方、ステップS1でCPを生成する可能性があるとして判定された場合は、次いでその節を含むプロセッサにCPがあるか否かを判定し(S2)、CPが無い場合はallocate命令の前にmark-prv-cp命令を生成し(S3)、CPが有る場合はallocate命令の前にmark-cur-cp命令を生成する(S5)。両者の場合について、カットオペレータ位置にはcut命令を生成する(S4)。

次に具体例に則して本実施例の装置の作用を説明する。

先ずカットオペレータを含む節が本体部実行中

に新たにCPを生成する可能性があり、且つ、その節を含むプロセッサがCPを生成している場合、即ち第2図のS3の場合について第3図に基づいて説明する。第3図(a)はプログラムの一例である。今節1を実行するため本体部のゴールPに対して節2を、ゴールQに対して節4を実行して、かつ節4の本体部のゴールSに対して節6を、ゴールTに対して節8を実行するとする。節4はカットオペレータを含みしかも永久変数を持つと仮定すると、コンパイルコードは第3図(b)の(1)~(11)の如くなる。(1)のtry-me-else命令でQに対するCPがローカルスタックに形成され、その後(2)のmark-prv-cp命令で前CPへのポインタがローカルスタックに格納される。そして(3)のallocate命令でENVが割付けられる。(5)、(6)の命令でゴールSが実行されると、同図(a)の節6が実行され、また(7)、(8)の命令でゴールTが実行されると、同図(a)の節8が実行される。節6、節8とも別解を持つためローカルスタックにCPが形成される。節8の実行時点でローカル

スタックは第3図(c)の如くなる。本図から明らかなようにENVは同図(a)の節1と、節4に対応するもの(それぞれENV(A), ENV(Q))が割付けられており、ENV(Q)の継続環境としてENV(A)へのポインタが入っている。また現在の(最新の)CPを指すポインタ・BレジスタはCP(T)を、現在のENVを指すポインタ・EレジスタはENV(Q)を指している。

ENV(Q)の手前には同図(b)の⑫mark-prv - cp命令で格納したバックトラックポイントがあり、CP(P)を指している。またENV(Q)には同図(a)の節4が永久変数を持つため、永久変数の値セルが格納されている。同図(b)で実行を進めて⑭のcut命令を実行する(これは同図(a)節4のカットオペレータを左から右へ通過することに相当する)とEレジスタのオフセット、即ちこの場合(E-1)番地からバックトラックポイントを取り出し、Bレジスタにロードする。この後、同図(b)の(10)(12)

図(d)のようになる。即ち、同図(c)でのENV(Q)が同図(d)ではCONT(Q)に変わっているのみである。EレジスタはやはりCONT(Q)を指している。従って同図(b)の(a)cut命令では(E-1)番地からバックトラックポイントをBレジスタにセットすることができる。

次にカットオペレータを含む節が本体部実行中に新たにCPを生成する可能性はあるが、その節を含むプロセッサがCPを生成していない場合、即ち第2図のS5の場合について第4図に基づき説明する。最初にカットオペレータを含む節が永久変数を持つ場合について説明する。第4図(a)に例としてあげたプログラムは、既に節を含むプロセッサがCPを生成している場合の説明で用いたものを一部変更して得たものである。即ち、カットオペレータを含む節5はプロセッサの最後に来ている。節1の実行に始まって、ゴールPに対して節2、ゴールQに対して節5を実行し、また節5の実行で、ゴールSに対して節6、ゴールT

の命令でゴールTが実行されるが、(11)のdeallocate命令でENV(Q)がローカルスタックから除かれ、その際、次に実行すべき命令として同図(a)のゴールRに対応するコードアドレス2が設定されるので、ゴールTの実行が成功するとゴールRに実行が移る。またゴールTの実行が失敗するとバックトラックが起こるが、既にBレジスタにはバックトラックポイントとしてCP(P)がセットされているため、CP(P)まで巻戻され、Pの代わりの節、同図(a)の節3が実行される。

以上はカットオペレータを含む節が永久変数を持つ場合であったが、次に永久変数を持たない場合を考える。同図(a)のプログラムで節4が永久変数を持たないと仮定する。この時もコンパイルコードは永久変数を持つ場合と同じで、同図(b)のままである。但し、同図(b)で3行目のallocate命令や11行目のdeallocate命令がENVを割付けるためではなく、CONTを割付ける目的を持つ。この結果ローカルスタックは第3

に対して節8を実行するものとする。節5のコンパイルコードは第4図(b)の(1)~(12)の命令に置換される。(1)のtrust - no - else fail命令で、節4と節5に対してあったCPが廃棄される。この時点で節2のCP、即ちCP(P)は生成されている。(2)のmark-cur - cp命令では現CPポインタの内容がローカルスタックのTOSに格納される。この後(3)のallocate命令でENVが割付けられる。命令(4)、(5)で節6が、命令(7)、(8)で節8が実行される。節6、節8ともローカルスタックにCPを生成する。従ってこの時点でのローカルスタックは第4図(c)の如くなる。現CPポインタはCP(T)を指し、現ENVポインタはENV(Q)を指している。そしてENV(Q)の手前にはCP(P)を指すバックトラックポイントが格納されている。従って第4図(b)で(9)のcut命令を実行すると、(E-1)番地からバックトラックポイントが取出され、Bレジスタに格納される。(10)、(12)の命令でゴールUが実行されるが、この実行が成功すると、節1のゴ

ールRに実行が移り、この実行が失敗すると、バックトラックが起り、CP(P)までCPが巻戻され、節2の代替の節である節3の実行が試される。

次にカットオペレータを含む節が永久変数を持たない場合、即ち第4図(a)の節5が永久変数を持たない場合について説明する。この場合のコンパイルコードも、永久変数を持つ場合と同じ、同図(b)の如くなるが、節(3)のallocate命令、節(11)のdeallocate命令の機能が、ENVではなくCONTをローカルスタックに生成、廃棄する処理に変わる。従って、この時のローカルスタックは同図(d)のようになるがこの場合もバックトラックポイントはレジスタのオフセットで取出すことができる。

代わってカットオペレータを含む節の実行中、明らかにCPの生成が無いと判定された場合について述べる。先ずその節に対応するCPがローカルスタックにある場合、即ち第2図のS7の場合を第5図に基づき説明する。同図(a)はプログ

ラム例である。節4は本体部にカットオペレータを含む。また変数Xは2つのゴールに現われるため、文献1の規則によれば永久変数であり、ENVの割付けが必要である。先ず文献1の規則通りにENVを割付けることを前提にして、説明を進める。第5図(a)で節1を実行するため、ゴールPに対して節2を、ゴールQに対して節4を実行するとする。節1でENV、節2でCP、節4でCPとENVが割付けられ、ローカルスタックは同図(c)の如くなる。さて、問題の節4は同図(b)の(1)~(9)の命令にコンパイルされる。(1)のtry-me-else命令でCP(Q)を生成し、(2)のallocate命令でENV(Q)を生成する。ここで注目すべきことは、allocate命令の前にmark-prv-cp命令やmark-cur-cp命令を置く必要はない。(4)、(5)の命令でvar(X)ゴールを実行し、次のカットオペレータを(6)のdeut命令で処理する。これは、Bレジスタ即ちCP(Q)の前のCP、即ちCP(P)へのポイントをBレジスタにロードする処理である。その後、ゴールX=Yを(9)の

unity X, Y命令(文献1のunify命令とは必ずしも厳密に対応していないことに注意)で実行する際、実行が成功した場合には節1のRに実行が移り、実行が失敗した場合にはバックトラックが発生して、Bレジスタが指すCP(P)まで巻戻され、節Pが実行されることになる。

次に当該節に対応するCPが無い場合について第6図に基づき説明する。第6図(a)はそのプログラム例で、第5図(a)と比較して節4と節5の位置が入れ替わっているだけである。この場合のコンパイルコードは第6図(b)の如くなる。(1)のtry-me-elseは節4の実行時に生成したCP(Q)を廃棄し、従ってローカルスタックは第6図(c)の如くなる。第5図(b)の場合と違い、(4)、(5)の命令でvar(X)を実行した後、カットオペレータ位置では何もしない。即ち、次のX=Yゴールを実行して失敗した後、カットオペレータを右から左へ通過しようとした時のバックトラックポイント、第6図(c)のCP(P)の位置がBレジスタに残っているからである。従

ってこの場合カットオペレータに対して何もする必要がない。既に述べたように節の本体部にあってサブゴールを起動しないで実行できる、従ってCPもENVも生成しないで実行できる述語は数値演算、論理演算、比較演算、その他の組み込み述語だが、インプリメンテーションによっても異なるため、予めこれらを分類分けして、コンパイル時にコンパイラが参照できるテーブルなどにまとめておく必要がある。

同様にこの方法に従えば、本体部にカットオペレータを含むか否かに拘らず、以上のように分類されたゴールのみで本体部が構成されている場合、その節の実行にあたっては、文献1の規則によれば永久変数を含みENVの割付けが必要な場合、或いはCONTの割付けが必要な場合も、これらのフレームをローカルスタックに割付けることなく高速に実行できる。この場合、ゴール間の引数の受渡しはレジスタを通して行なう。この方法によれば、第5図(a)の節4は第7図の如く、又、第6図(a)の節5は第8図の如くコンパイルし

て実行することができる。

〔発明の効果〕

以上述べたように、本発明によれば、PROLOG言語のコンパイル時に、本体部のゴールを判定して、ローカルスタックにバックトラックポイントを割付ける必要のないものは、その処理を省くようにしているので、この割付けのためのメモリアクセスを省略でき、処理のスピードアップを図ることができる。また、従来の規則によらない新しい規則でローカルスタックへのCONTフレームの割付けを行なうことで、バックトラックポイントをローカルスタックに格納した場合、必ず現ENVポインタのオフセットで取出せるので、特にカットオペレータを含むPROLOGプログラムを高速に処理できる。

4. 図面の簡単な説明

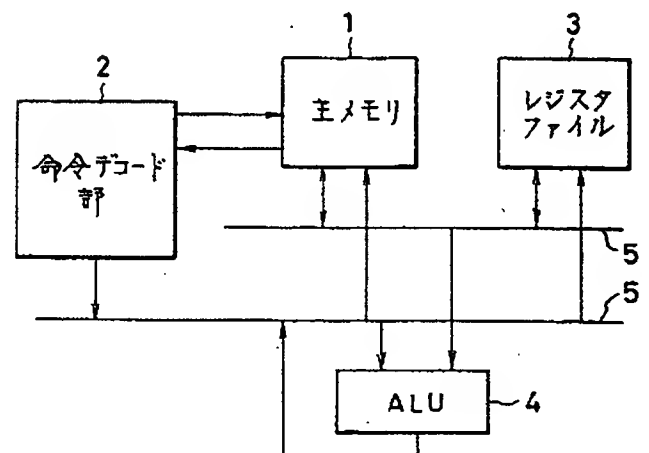
第1図～第8図は本発明の一実施例を説明するための図で、第1図は本発明の一実施例に係るPROLOG処理装置の構成を示すブロック図、第2図は同処理装置におけるコンパイラのカット

オペレータの処理フローを示す流れ図、第3図は第2図のステップS3の処理例を示す図で、同図(a)はプログラム例、同図(b)は同プログラムに対応するマシン命令、同図(c)及び(d)はローカルスタックの様子をそれぞれ示す図、第4図は第2図のステップS5の処理例を示す図で、同図(a)はプログラム例、同図(b)は同プログラムに対応するマシン命令、同図(c)及び(d)はローカルスタックの様子をそれぞれ示す図、第5図は第2図のステップS7の処理例を示す図で、同図(a)はプログラム例、同図(b)は同プログラムに対応するマシン命令、同図(c)はローカルスタックの様子をそれぞれ示す図、第6図は第2図のステップS6でCPが無い場合の処理例を示す図で、同図(a)はプログラム例、同図(b)は同プログラムに対応するマシン命令、同図(c)はローカルスタックの様子をそれぞれ示す図、第7図及び第8図はカットオペレータを含まない節を処理するためのマシン命令をそれぞれ示す図、第9図は環境フレームの構成図、第

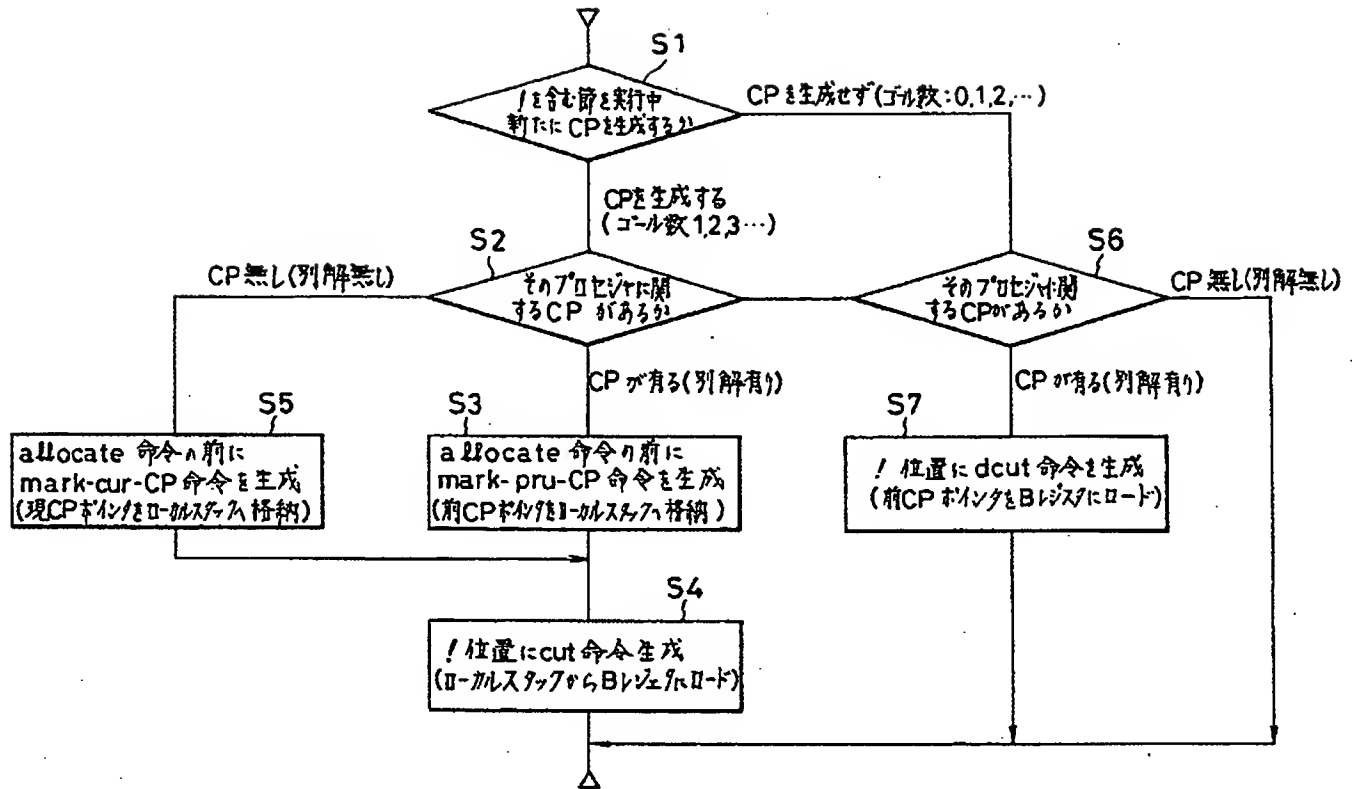
10図は選択点フレームの構成図、第11図は従来のPROLOG処理装置でプログラムをマシン命令に展開した図、第12図はプログラム例とローカルスタックの様子とを示す図、第13図はカットオペレータを含むプログラム例とローカルスタックの様子とを示す図である。

1…主メモリ、2…命令デコード部、3…レジスタファイル、4…ALU、5…データバス。

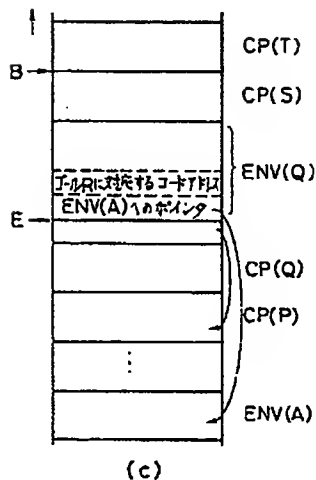
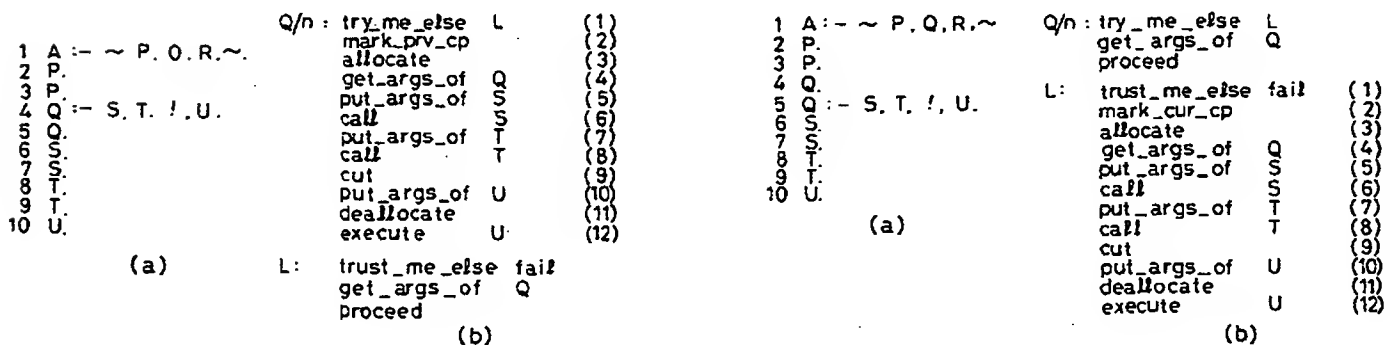
出願人代理人 弁理士 鈴江武彦



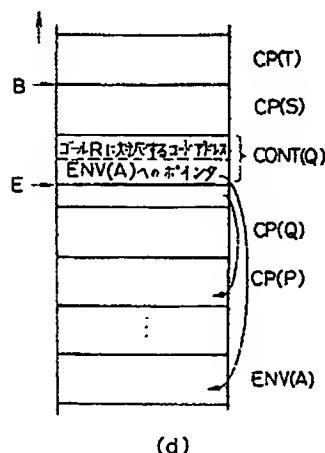
第 1 図



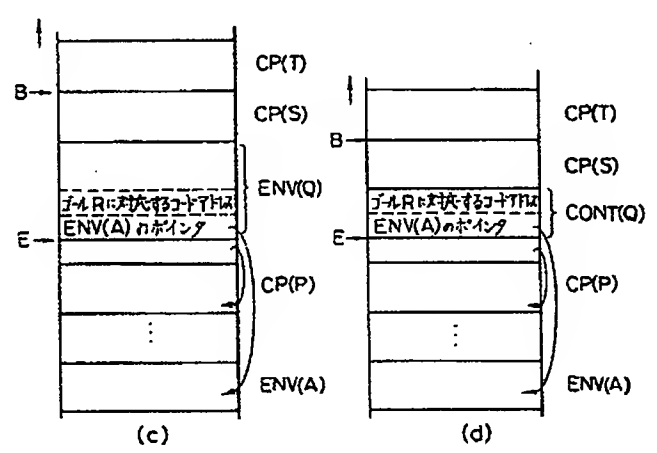
第 2 図



第 3 図



(d)



(c)

第 4 図

```

1 A :- ~ P, Q, R, ~.
2 P.
3 P.
4 Q :- var(X), !, X=Y.
5 R.
6 R.
7 R.

```

(a)

```

Q/n: try_me_else      (1)
      allocate        (2)
      get_args_of     Q      (3)
      put_args_of     var    (4)
      switch_on_term  fail fail fail (5)
      dcut            (6)
      deallocate      (7)
      unify           X,Y    (8)
      proceed         (9)

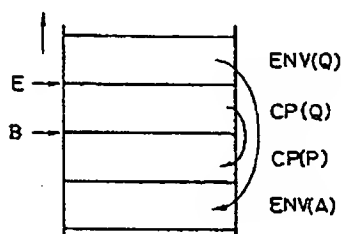
```

```

L:  trust_me_else fail
      get_args_of  Q
      proceed

```

(b)



(c)

第 5 図

```

1 A :- ~ P, Q, R, ~.
2 P.
3 P.
4 Q :- var(X), !, X=Y.
5 R.
6 R.
7 R.

```

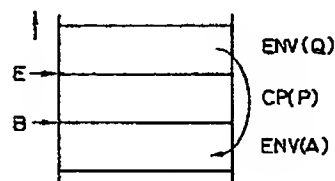
(a)

```

Q/n: try_me_else L
      get_args_of Q
      proceed
L:  trust_me_else fail      (1)
      allocate        (2)
      get_args_of     Q      (3)
      put_args_of     var    (4)
      switch_on_term  fail fail fail (5)
      deallocate      (6)
      unify           X,Y    (7)
      proceed         (8)

```

(b)



(c)

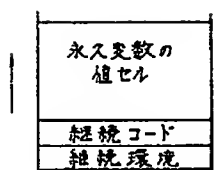
第 6 図

```

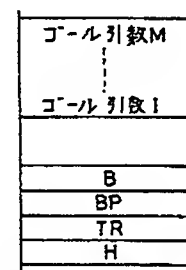
Q/n: try_me_else L
      get_args_of Q
      put_args_of var
      switch_on_term fail fail fail
      dcut
      unify X,Y
      proceed

```

第 7 図



第 9 図



第 10 図

```

L:  trust_me_else fail
      get_args_of  Q
      put_args_of  var
      switch_on_term fail fail fail
      unify       X,Y
      proceed

```

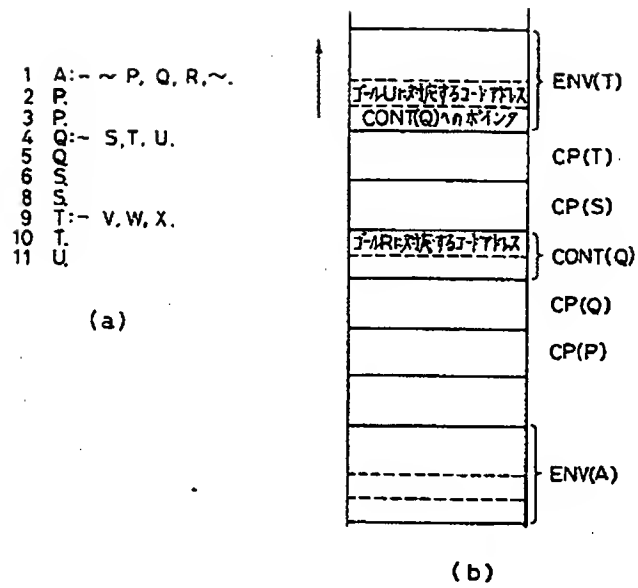
第 8 図

```

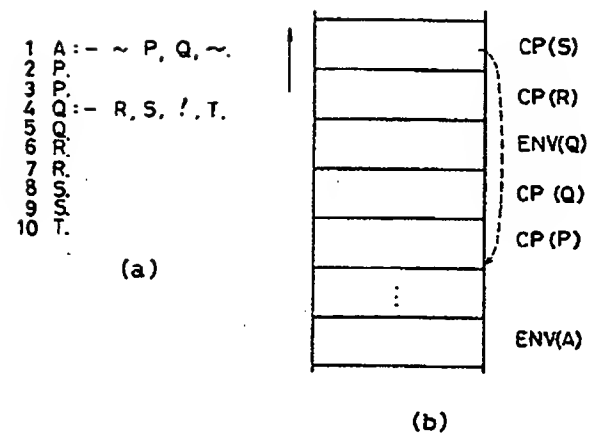
1 allocate
2 get_args_of A
3 put_args_of B
4 call      B
5 put_args_of C
6 call      C
7 put_args_of D
8 deallocate
9 execute   D

```

第 11 図



第 12 図



第 13 図